ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatik I - D-ITET

Wintersemester 2003/2004

Prof. Markus Gross, Bruno Heidelberger, Richard Keiser, Nicky Kern, Edouard Lamboray, Christoph Niederberger, Tim Weyrich, Felix Eberhard, Manuel Graber, Nathalie Kellenberger, Marcel Kessler, Lior Wehrli

Das Miniprojekt

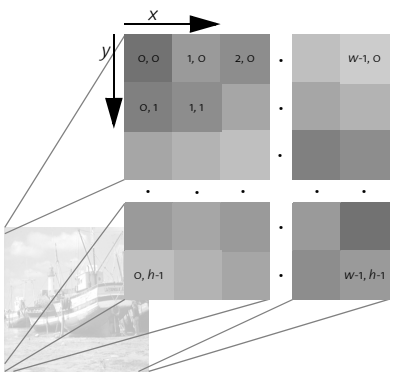
Ausgabe:	5. November 2003
Abgabe:	28. und 30. Januar 2004
Autor:	Tim Weyrich

1. Allgemeine Beschreibung

Das Miniprojekt ist eine Programmieraufgabe, die während des laufenden Semesters zu bearbeiten ist. Es besteht aus einem Pflichtteil, dessen Bearbeitung 110 Punkte für das Testat bringt. Daneben darf das Miniprojekt beliebig über die Aufgabenstellung hinaus erweitert werden — die Erweiterungen nehmen an einem Wettbewerb teil, und das gelungenste Programm wird prämiert.
Im Rahmen des Miniprojekts sollen einfache Verfahren aus der Computergraphik in ein kleines Malprogramm integriert werden. Dazu stellen wir euch ein kleines Rahmenprogramm zur Verfügung, mit dem ihr Bilder laden, speichern und anzeigen könnt. Das Rahmenprogramm und weitere Inforationen zum Projekt findet ihr auf der Informatik I Homepage unter *http://graphics .ethz.ch/info1/*.
Wer noch nie zuvor programmiert hat, dem mögen die hier vorgestellten Aufgaben auf den ersten Blick etwas schwierig erscheinen. Wir denken aber, dass spätestens nach Weihnachten jeder in der Lage sein sollte, die Aufgaben zu lösen. Für die Bearbeitung sind keine besonderen Vorkenntnisse aus der Computergraphik notwendig. Ihr solltet lediglich wissen, dass es Farb- und Grauwertbilder gibt und dass Bilder in der Computergraphik meistens aus kleinen quadratischen Bildelementen, sogenannten *Pixeln*, zusammengesetzt sind. Dieser Zusammenhang sei hier noch einmal formalisiert dargestellt:

Grauwertbilder

Ein Grauwertbild ist eine matrixartige Anordnung von *Pixeln*, denen ein *Grauwert* (oder besser eine *Intensität*) zwischen 0 und 1 zugeordnet ist. 0 steht für Schwarz, 1 für Weiss[1]. Werte dazwischen stehen für unterschiedliche Abstufungen von Grau.



Die Pixel werden in einem *xy*-Koordinatensystem adressiert, dessen Ursprung in der linken oberen Ecke des Bildes liegt. Die *x*-Koordinate wächst nach rechts, die *y*-Koordinate nach unten an. In einem Bild der Breite *w* und der Höhe *h* hat das Pixel in der oberen linken Ecke die Koordinaten (0, 0), das in der rechten unteren Ecke (*w*-1, *h*-1).
Den Grauwert eines Pixels an der Stelle (*x*, *y*) bezeichnen wir mit *I*(*x*, *y*).

1. In der Praxis speichert man in den Pixeln keine Werte zwischen 0 und 1, da die Grauwerte jeweils in einem `unsigned char` abgespeichert werden. Um den Wertebereich voll auszuschöpfen, speichert man hier Werte zwischen 0 und 255. Am Prinzip ändert sich allerdings nichts, und wir sprechen weiterhin von Intensitäten zwischen 0 und 1.

Details zur Art und Weise, wie das gestellte Rahmenprogramm Grauwertbilder im Speicher ablegt, findet ihr im Netz unter http://graphics.ethz.ch/~weyrich/info-i/painter/pgdocImageGray.html.

2. Aufgaben

Der Pflichtteil umfasst drei Operationen aus der Bildbearbeitung. Alle Teilaufgaben arbeiten auf einem Grauwertbild. Mithilfe dieser Aufgaben sind bis zu 110 Punkte zu erreichen. Die Punktaufteilung ist bei den jeweiligen Teilaufgaben vermerkt.

Aufgabe 1: Linien Zeichnen **20 Punkte**

Durch einen Tastendruck soll das Programm in einen 'Linien-Modus' versetzt werden, in dem man bei gedrückter linker Maustaste Linien auf das Bild malen kann. Eine solche Operation sollte euch von vielen Malprogrammen her bekannt sein.

Für diese Aufgabe hält das zur Verfügung gestellte Rahmenprogramm Funktionen bereit, um auf Mausklicks und -bewegungen zu reagieren. Zur Steuerung der Maloperationen müsst ihr diese Mausaktionen entsprechend auswerten.

Der Ablauf einer Mal-Operation im 'Linien-Modus' sollte folgendermassen aussehen: Sobald die linke Maustaste gedrückt wird, wird die erste Ecke der Linie gespeichert. Solange nun die Maus bei gedrückter Taste bewegt wird, wird die Linie, die durch die erste Ecke und die aktuelle Mausposition definiert ist (siehe Abbildung 1 (a)), speziell hervorgehoben. Wenn die Maustaste losgelassen wird, wird die zur Zeit hervorgehobene Linie mit einem konstanten Grauwert ausgefüllt (siehe Abbildung 1 (b)).

Das Hervorheben einer Linie bei gedrückter Maustaste soll durch *Inversion der Grauwerte* implementiert werden. Der Grauwert jedes Pixels $I(x, y)$ wird dabei durch sein Inverses $1 - I(x, y)$ ersetzt. Man kann also durch zweimalige Inversion den ursprünglichen Wert des Pixels, bzw. den ursprünglichen Bildbereich, wieder herstellen.

Das Rahmenprogramm stellt eine Funktion zur Verfügung, um Linien im Bild zu invertieren, bzw. mit einem einheitlichen Grauwert zu zeichnen.

(a)  (b) 

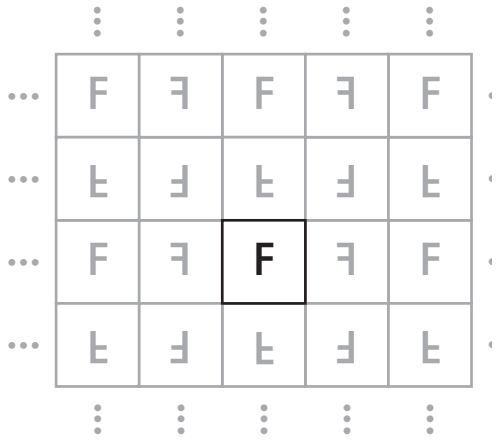
Abbildung 1: Linien Zeichnen: *(a)* Der Endpunkt einer invertierten Linie wird dem Mauszeiger bei gedrückter linker Taste nachgeführt. *(b)* Nachdem die Taste losgelassen wurde, wird die Linie mit einem einheitlichen Grauwert gezeichnet.

Aufgabe 2: Kaleidoskop **45 Punkte**

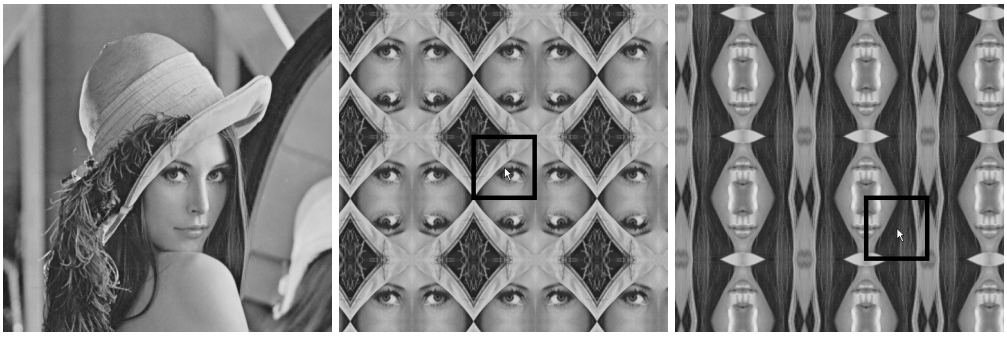
In der zweiten Aufgabe ist eine einfach Form eines *Kaleidoskops* zu programmieren: Bei

gedrückter Maustaste soll ein gedachtes quadratisches „Fenster“ über dem Bild verschoben werden. Innerhalb dieses Fensters ist das Bild zu sehen, wie es zum Zeitpunkt aussah, als die Maustaste gedrückt wurde. Alles was ausserhalb des Quadrats liegt, soll durch gespiegelte Versionen des Inneren des Quadrats ersetzt werden.

Die Spiegelungen sollen so erfolgen, als wären die Seiten des Quadrats Spiegel, an denen der Inhalt des Quadrats (mehrfach) „umgeklappt“ wird. Schematisch sieht die dadurch entstehende „Kachelung“ wie in der nebenstehender Abbildung aus. Der unter dem Fenster sichtbare Bildausschnitt ist mit einem schwarzen "F" angedeutet. Die Anordnung der durch Spiegelungen erzeugten Kopien ist grau wiedergegeben



Wird die Maus bei gedrückt gehaltener Maustaste verschoben, soll sich das gedachte Quadrat mit der Maus mitbewegen. Im Inneren des Quadrats sei dabei stets das ursprüngliche Bild zu sehen, als ob man ein Fenster über dem darunterliegende Originalbild verschieben würde. Der umliegende Bildbereich soll dabei immer wieder durch gespiegelte Varianten des Quadratinhalts ersetzt werden. Abbildung 2 zeigt drei Schnappschüsse dieses Ablaufs.



(a) (b) (c)

Abbildung 2: Screenshots des Kaleidoskop-Effekts. *(a)* Das urspruengliche Bild, kurz bevor die Maustaste gedrückt wird. *(b)* Die linke Maustaste wurde gedrückt, woraufhin die Kachel unter der Maus in gespiegelten Varianten über das Bild verteilt wird. Die gedachte Kachel ist umrandet hervorgehoben. Im Programm braucht eine solche Hervorhebung nicht zu erfolgen. *(c)* Die Maus wurde bei gedrückter Maustaste verschoben, und wieder ist der darunterliegende Bildbereich in das umliegende Bild gespiegelt worden. Man beachte, dass der vervielfaeltigte Ausschnitt aus dem urspruenglichen Bild in (a), nicht aus (b) stammt.

Ein Tipp zur Implementierung: Schreibt eine Funktion, die euch zu einem Pixel (x, y) im Kaleidoskop-Effekt-Bild berechnet, von welcher ursprünglichen Position (u, v) des Originalbilds es stammt. Dabei könnte u.U. der Modulo-Operator nützlich sein.

Und noch ein letzter Tipp zur Optimierung: Im gegebenen Rahmenprogramm wird das Neuzeichnen des Kaleidoskops bei Mausbewegungen flüssiger ausgeführt, wenn ihr einen Weg findet, das Neuzeichnen in `idleCB()` durchzuführen.

Aufgabe 3: Abbildungen und Bildinterpolation **45 Punkte**

Diese Aufgabe beschäftigt sich mit *Abbildungen* von Pixelbildern. Im Bereich der Bildbearbeitung kann eine Abbildung als Vorgang verstanden werden, der Pixelpositionen aus

einem *Quellbild* Positionen in einem *Zielbild* zuordnet. Bereits Aufgabe 2 hat eine solche Abbildung enthalten: Hier wurde der Inhalt des Quadrats um den Mauszeiger (mehrfach) auf umliegende Bildbereiche abgebildet. Formal lässt sich eine Abbildung schreiben als

$$(x', y') = T(\,(x, y)\,)\,. \tag{1}$$

oder, alternativ:

$$\begin{aligned} T\colon\quad & x' = T_x(x) \\ & y' = T_y(y)\,. \end{aligned} \tag{2}$$

Beide Schreibweisen besagen, dass die Abbildung T die Pixelposition (x, y) aus dem Quellbild auf (x', y') im Zielbild abbildet. T kann auch als eine Funktion verstanden werden, die x und y als Argumente nimmt und x' und y' zurückgibt. Eine Abbildung lässt sich auf ein Bild *anwenden*, indem Grauwerte entsprechend der Abbildung vom Quellbild in das Zielbild kopiert werden:

$$I(x', y') = I(x, y), \qquad \text{bzw.} \tag{3}$$

$$I(T(\,(x, y)\,)) = I(x, y)\,. \tag{4}$$

Wie implementiert man nun das Anwenden einer Abbildung auf ein Bild? Es wäre naheliegend, gemäss Gleichung (4) über das Quellbild zu laufen und an jeder Pixelposition (x, y) den Grauwert auszulesen, um ihn im Quellbild an die Stelle $(x', y') = T((x, y))$ zu schreiben.
Dies hätte aber Nachteile. So würden zum Beispiel bei Anwendung der Abbildung

$$(x', y') = S(\,(x, y)\,) = (2 \cdot x, 2 \cdot y) \tag{5}$$

Lücken im Zielbild auftreten. (Warum? Was macht diese Abbildung anschaulich gesprochen mit dem Quellbild?). Ein weiteres Problem tritt auf, wenn die Abbildung (x, y) auf *nicht ganzzahlige* (x', y') abbildet. Dann stellt sich die Frage, in welches Pixel der Grauwert überhaupt geschrieben werden soll.
In der Praxis geht man daher einen anderen Weg. Um sicherzustellen, dass im Zielbild jedes Pixel eindeutig mit einem Grauwert belegt wird, schreitet man alle Pixel im *Ziel*bild ab und bestimmt dort für jedes (x', y'), von welcher (x, y) Position es stammt. Von dort nimmt man dann den Grauwert $I(x, y)$, um das Zielpixel einzufärben.
Zu einem gegebenen (x', y') das entsprechende (x, y) zu finden entspricht der Anwendung der *inversen Abbildung*. Die inverse Abbildung ist durch ein „$^{-1}$“ gekennzeichnet. Die Inverse zu T wird zum Beispiel mit T^{-1} bezeichnet. Es gilt

$$(x, y) = T^{-1}(\,(x', y')\,) \tag{6}$$

und insbesondere

$$T^{-1}(T(\,(x, y)\,)) = (x, y)\,. \tag{7}$$

Das Finden einer inversen Abbildung T^{-1} zu einem gegebenen T (und die Frage, ob so ein T^{-1} überhaupt *existiert*) ist ein häufiges Problem in der Mathematik, soll hier aber nicht weiter behandelt werden. In unserem Fall nehmen wir an, dass T^{-1} bekannt sei. Tatsächlich benötigen wir im Folgenden nur noch T^{-1}. T wird für die Abbildung eines Pixelbilds

nicht mehr benötigt.
Zusammenfassend sei also festgehalten: Um eine Abbildung T auf ein Quellbild anzuwenden, muss jedes Pixel (x', y') im Zielbild mit $I(T^{-1}((x', y')))$ belegt werden.

Rotation eines Pixelbilds

Eine häufig benötigte Abbildung ist die Rotation. Eine Abbildung R_φ die alle Punkte eines Bilds um einen Punkt (x_O, y_O) um den Winkel φ rotiert, lässt sich wie folgt beschreiben:

$$\begin{aligned} R_\varphi: \quad x' &= x_O + (x - x_O) * \cos(\varphi) - (y - y_O) * \sin(\varphi) \\ y' &= y_O + (x - x_O) * \sin(\varphi) + (y - y_O) * \cos(\varphi) . \end{aligned} \tag{8}$$

φ dreht hier übrigens im Gegensatz zur üblichen Konvention im Uhrzeigersinn, da Pixelbilder im allgemeinen ein *Linkskoordinatensystem* besitzen.
Für die Anwendung auf ein Pixelbild benötigen wir wieder die inverse Abbildung. Bei einer Rotation ist das die gleiche Rotation, nur um den negativen Winkel. Es gilt also: $R_\varphi^{-1} = R_{-\varphi}$.
Erweitert nun das Rahmenprogramm, so dass das aktuelle Bild auf Tastendruck um 15° im Uhrzeigersinn um den Punkt (265, 265) gedreht wird. Beachtet dabei, dass `sin()` und `cos()` aus `math.h` im Bogenmass rechnen, Winkel also statt von 0° bis 360° von 0 bis 2π verlaufen.
Bei der Anwendung von R_φ^{-1} können selbstverständlich auch nicht ganzzahlige x und y entstehen. Doch wie soll man verfahren, wenn die Koordinaten nicht ganzzahlig sind, man also eigentlich "zwischen" den Pixelmittelpunkten einen Grauwert lesen möchte?
Die Aufgabe, Zwischenwerte zu berechnen, wo nur einzelne Werte an diskreten Positionen (den Pixelmittelpunkten) gegeben sind, nennt man *Interpolation*.
In unserem Fall wäre eine einfache Lösung, die x/y Koordinaten jeweils zu den nächsten ganzen Zahlen zu runden. Das hiesse, immer das nächstgelegene Pixel zu wählen. Diese Interpolationsstrategie nennt man *Nearest Neighbour*.
Beobachtet, was passiert, wenn Ihr in eurem Rotationscode die Nearest-Neighbour-Interpolation verwendet. Insbesondere, wenn die Rotation mehrfach hintereinander ausgeführt wird, sollten dabei unerwünschte Ergebnisse entstehen.
Die Abbildung kann erheblich verbessert werden, wenn eine andere Interpolationsstrategie verwendet wird. Eine häufig verwendete Interpolation ist die *bilineare Interpolation*. Dabei wird $I(x, y)$ für beliebe x und y als

$$\begin{aligned} I(x,y) = {} & (1-\lambda_1)(1-\lambda_2) \cdot I(\lfloor x \rfloor, \lfloor y \rfloor) \\ & + \lambda_1(1-\lambda_2) \cdot I(\lfloor x+1 \rfloor, \lfloor y \rfloor) \\ & + (1-\lambda_1)\lambda_2 \cdot I(\lfloor x \rfloor, \lfloor y+1 \rfloor) \\ & + \lambda_1\lambda_2 \cdot I(\lfloor x+1 \rfloor, \lfloor y+1 \rfloor) \end{aligned}$$

mit

$$\begin{aligned} \lambda_1 &= x - \lfloor x \rfloor \\ \lambda_2 &= y - \lfloor y \rfloor \end{aligned}$$

definiert ($\lfloor x \rfloor$ bezeichnet die `floor()`-Operation, die zur nächsten ganzen Zahl abrundet).
Die bilineare Interpolation kombiniert $I(x, y)$ für nicht ganzzahlige x und y aus den Grauwerten der vier nächstgelegenen Pixel. Wenn x und y ganzzahlig sind, geht nur das an dieser Stelle befindliche Pixel ein, wie sich anhand der Formel leicht überprüfen lässt.
Implementiert die Rotation auch unter Verwendung bilinearer Interpolation und vergleicht das Ergebnis mit der Nearest-Neighbour-Interpolation.

Bei beiden Interpolationsverfahren kann es vorkommen, dass ihr versucht Pixel ausserhalb des Bildes auszulesen. Überlegt euch, wie in diesem Fall am sinnvollsten zu verfahren ist.
Für die Implementierung der Rotation mit Nearest-Neighbour-Interpolation erhaltet ihr 10 Punkte. Die bilineare Interpolation bringt weitere 15 Punkte.

Interaktive Achsenspiegelung

Als letzter Teil dieser Aufgabe ist eine (interaktive) Achsenspiegelung zu implementieren. Wie in Aufgabe 1 soll der Benutzer eine Linie über dem Bild aufziehen. Lässt er die Maustaste los wird allerdings keine grauwertige Linie gezeichnet, sondern es sollen alle Pixel, die sich rechts von der aufgezogenen Linie befinden, an der Gerade durch Anfangs- und Endpunkt der Linie gespiegelt werden. Abbildung 4 zeigt den entsprechenden Programmablauf.

(a) (b)

Abbildung 3: Screenshots der interaktiven Achsenspiegelung. *(a)* Analog zu Aufgabe 1 wurde mit der Maus eine Linie aufgezogen. Die linke Maustaste ist noch gehalten. *(b)* Nachdem die Maustaste losgelassen wurde, ist der Bildbereich rechts von der aufgezogenen Linie in den gegebüberliegenden Bildbereich gespiegelt worden.

Zur dieser Aufgabe geben wir eine kleine Hilfestellung. Wir verwenden dabei die Vektorschreibweise für Pixelpositionen. Statt (x, y) schreiben wir:

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \end{bmatrix} .$$

Vektoren seien hier durch Fettdruck gekennzeichnet. Die beiden Komponenten eines Vektors $\boldsymbol{p}$ werden mit p_1 und p_2 bezeichnet. Für Vektoren gelten die bekannten Rechenregeln:

$$\boldsymbol{a} + \boldsymbol{b} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \end{bmatrix}, \qquad s\boldsymbol{a} = \begin{bmatrix} sa_1 \\ sa_2 \end{bmatrix}, \qquad \boldsymbol{a} \bullet \boldsymbol{b} = a_1 b_1 + a_2 b_2 ,$$

$$\|\boldsymbol{a}\| = \sqrt{a_1^2 + a_2^2} .$$

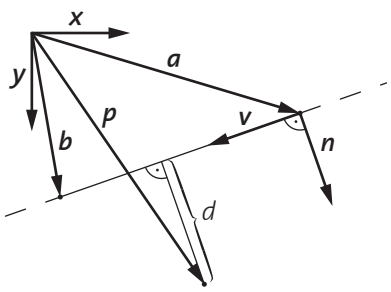
Angenommen, die mit der Maus aufgezogene Linie verläuft von $\boldsymbol{a}$ nach $\boldsymbol{b}$. Dann be-

schreibt diese eine Gerade

$$g: \boldsymbol{a} + s\boldsymbol{v} \qquad \text{mit} \qquad \boldsymbol{v} = \frac{\boldsymbol{b} - \boldsymbol{a}}{\|\boldsymbol{b} - \boldsymbol{a}\|} .$$

(Vergleiche auch die Skizze in nebenstehender Abbildung.)
Senkrecht auf dieser Gerade steht der Normalenvektor

$$\boldsymbol{n} = \begin{bmatrix} v_2 \\ -v_1 \end{bmatrix} .$$



Es gilt $\|\boldsymbol{n}\| = 1$. Für die gegebene Aufgabe muss festgestellt werden, ob eine Pixelposition $\boldsymbol{p}$ links oder rechts (in der Richtung von $\boldsymbol{v}$ gesehen) von der Geraden liegt. Dazu kann man die Entfernung d von $\boldsymbol{p}$ zu g bestimmen:

$$d = (\boldsymbol{p} - \boldsymbol{a}) \bullet \boldsymbol{n} .$$

Je nach Vorzeichen von d befindet sich $\boldsymbol{p}$ links oder rechts von g.
Mithilfe dieser Formeln sollte es euch möglich sein, die interaktive Spiegelung zu programmieren. Für die interaktive Achsenspiegelung erhaltet ihr 20 Punkte.

3. Das vorgegebene Rahmenprogramm

Damit ihr euch bei der Bearbeitung der Aufgabe möglichst ganz auf den Bildbearbeitungsteil konzentrieren könnt, stellen wir euch ein kleines Rahmenprogramm zur Verfügung, das euch bereits einige Arbeit abnimmt. Das Rahmenprogramm bietet

- die permanente Anzeige eines Grauwert- oder Farbbilds.
- Laden und Speichern von Grauwert- und Farbbildern im TIFF-Format.
- sogenannte Callback-Funktionen, damit ihr auf Tastendrücke und Mausaktionen reagieren könnt.

Eine ausführliche Beschreibung des Rahmenprogramms findet ihr unter http://graphics.ethz.ch/~weyrich/info-i/painter/. *Dort solltet ihr euch als Erstes die Sektion 'Related Pages' anschauen.* Die Dokumentation des Rahmenprogramms ist auf Englisch gehalten.
Der Quellcode zum Rahmenprogramm befindet sich in http://graphics.ethz.ch/~weyrich/info-i/painter-framewk.tar.gz.
Prinzipiell darf das vorgegebene Framework natürlich verändert werden. Die Lösung der Aufgaben ist allerdings auch ohne solche Eingriffe möglich.
Die Unterstützung von Farbbildern ist ein Angebot für diejenigen, die es gerne etwas spektakulärer haben ;-) Wenn es euch aber nur um das Bearbeiten der eigentlichen Aufgabenstellung geht, empfehlen wir euch, bei Grauwertbildern zu bleiben, um euch das Leben nicht allzu schwer zu machen, da mit der Verwendung von Farbbildern weitere Herausforderungen auf euch zukommen...

4. Abgabe des Pflichtteils

Die in diesem Dokument vorgestellten testatrelevanten Aufgaben sind beim jeweiligen Assistenten abzugeben. Die Abgaben erfolgen über zwei Tage verteilt am 28. und 30. Januar 2004, jeweils von 13:00 bis 15:00 Uhr im Rechnerraum ETZ D96. Euer Assistent wird euch seinen Abgabetermin noch bekannt geben. Leider können aus technischen Gründen nicht alle Abgaben am gleichen Tag stattfinden.
Die Projektabnahme durch den Assistenten findet direkt an den Rechnern im Sun-Pool statt. Es werden nur Aufgaben bewertet, die als lauffähiges Programm vor Ort vorgeführt werden können. Daher ist es insbesondere für Leute, die die Aufgaben am heimischen Linux-PC gelöst haben, *unbedingt notwendig*, regelmässig zu überprüfen, ob ihr Programm auch noch *auf den Sun Computern des Rechner-Pools lauffähig* sind. Auch

wenn sich die Umgebungen unter Verwendung des GNU C-Compilers weitestgehend gleichen, muss immer mit Kompatibilitätsproblemen gerechnet werden. Mehr zum Thema 'Entwickeln auf Fremdplattformen' findet ihr in der online-Dokumentation zum Rahmenprogramm.

5. Wettbewerb

Mit dem Miniprojekt ist auch ein Programmierwettbewerb verbunden. Jeder ist herzlich eingeladen, besonders gelungene Erweiterungen am Miniprojekt einzureichen.

Was bewertet wird

Am Wettbewerb können Projektabgaben teilnehmen, die über das Pflichtprogramm hinaus interessante Erweiterungen an ihrem Programm vorgenommen haben. Dabei werden vor allem gewitzte Aktionen auf dem Pixelbild bewertet. Lasst eurer Phantasie freien Lauf — professionelle Mal- und Bildbearbeitungsprogramme bieten schon einige Funktionen, von denen ihr euch inspirieren lassen könnt. Aber vielleicht fällt euch auch noch etwas Anderes ein, das man auf dem Pixelbild machen kann? Erlaubt ist, was gefällt...
Keinen besonderen Augenmerk bei der Bewertung werden wir auf Dinge wie besonders aufwendige Benutzeroberflächen, Soundeffekte, Web-Interface, Horoskop-Generator, online-Buchungssystem, etc. legen. — Die Arena ist das Pixelbild!
Dennoch werden wir es wohlwollend bewerten, wenn das Programm ein gutes objektorientiertes Design aufweist. Dabei kann auch ruhig das zur Verfügung gestellte Rahmenprogramm, das ja komplett in C (ohne ++) geschrieben ist, für den OO-Bedarf umgestrickt werden...

Dokumentation

Bei der Abgabe muss *unbedingt* eine Dokumentation eures Programms mitgeschickt werden. Das kann als getrennt abgegebenes Textdokument oder als README-Datei in eurem Programmverzeichnis erfolgen. Bitte schickt uns keine Word-Dokumente. Akzeptiert werden reine Textdateien, Postscript, PDF und HTML.
Aus der Dokumentation muss hervorgehen, welche Erweiterungen euer Programm gegenüber dem Pflichtteil hat. Funktionalität, die nicht in der Dokumentation erwähnt ist, können wir nicht bewerten. Darüberhinaus muss die Benutzersteuerung aus der Dokumentation klar hervorgehen.

Abgabe der Wettbewerbsbeiträge

Die Abgabe der Wettbewerbsbeiträge muss per Email bis spätestens Donnerstag, den 29. Januar, 18:00 Uhr an 'Tim Weyrich <weyrich@inf.ethz.ch>' erfolgen. Abgaben, die nach diesem Zeitpunkt eintreffen, werden nicht berücksichtigt.
Die Mail muss das Subject 'Abgabe' besitzen und die Namen aller Beteiligten (maximal 2 Personen) enthalten. Das Projekt muss der Mail als .tar.gz-Archiv anhängen und sich auf den tardis-Suns entpackt per 'gmake all' kompilieren lassen.
Bitte habt Verständnis, dass wir nur Einsendungen bewerten können, die diese Voraussetzungen erfüllen.

Preisverleihung

Auf die Gewinner des Wettbewerbs wartet ein attraktiver Preis. Die Bekanntgabe der Gewinner und die Preisverleihung finden in der letzten Übung des Semesters, Montag, den 4. Februar 2004, statt.
Der Rechtsweg ist ausgeschlossen.